Name: Vedant Patel ID: 030855576

Optimization of Edit distance Algorithm using SIMD

Report

Reference of a program: https://www.geeksforgeeks.org/edit-distance-dp-5/

Introduction:

Given two strings **str1** and **str2** of length **M** and **N** respectively and below operations that can be performed on **str1**. Find the minimum number of edits (operations) to convert '**str1**' into '**str2**'.

- **Operation 1 (INSERT)**: Insert any character before or after any index of **str1**
- **Operation 2 (REMOVE):** Remove a character of **str1**
- **Operation 3 (Replace):** Replace a character at any index of **str1** with some other character.

Code and Functionality:

Functionality:

The original code performs edit distance computation using dynamic programming, where each cell in the matrix is updated based on the neighboring cells. This approach has time complexity O(m * n), where m and n are the lengths of the input strings.

File name: Edit_dis.cpp

Original Program:

Program flow:

1. **Include Header Files**:
   - The program includes necessary header files such as **<bits/stdc++.h>**, **<iostream>**, **<string>**, **<vector>**, **<random>**, and **<chrono>**. These header files provide functions and classes needed for various operations.
2. **Solution Class Definition**:
   - The **Solution** class is defined, encapsulating the functionalities related to string generation and edit distance calculation.

3. **String Generation Function (generateLargeString)**:
    - This function generates a random string of a specified length using lowercase English alphabets.
4. **Edit Distance Calculation Function (editDistance)**:
    - This function calculates the edit distance between two strings using dynamic programming.
5. **Main Function (main)**:
    - In the **main** function:
        - An instance of the **Solution** class is created.
        - Two large random strings of different lengths are generated using the **generateLargeString** function.
        - The lengths of the generated strings are printed to the console.
        - The current time is recorded before and after the **editDistance** function call to measure the execution time.
        - The edit distance between the two generated strings is calculated using the **editDistance** function.
        - The calculated edit distance and the execution time are printed to the console.
6. **Execution**:
    - When the program is executed:
        - Random strings are generated.
        - The edit distance between the generated strings is calculated.
        - The lengths of the strings, the edit distance, and the execution time are printed to the console.
7. **Termination**:
    - After execution, the program terminates, and the control returns to the operating system.

Optimized program:

Program flow:

This C++ program calculates the edit distance between two randomly generated large strings using dynamic programming with SIMD (Single Instruction, Multiple Data) optimization. Here is the break down the code and explain its functionality, program flow, and how SIMD is used to accelerate the computation.

1. **Header Includes:** The code includes necessary header files for input-output operations, string manipulation, random number generation, timing, vector manipulation, and SIMD intrinsics.

2. **Solution Class:**
    - The generateLargeString() method generates a random string of a specified length using characters from the English alphabet.
    - The editDistance() method calculates the edit distance between two strings using dynamic programming with SIMD optimization.
3. **Main Function:**
    - It creates an instance of the Solution class.
    - Generates two large random strings of different lengths using the generateLargeString() method.
    - Calculates the edit distance between the generated strings using the editDistance() method.
    - Measures the execution time using chrono::steady_clock to profile the performance.
    - Prints the lengths of the generated strings and the calculated execution time.
4. **SIMD Optimization:**
    - The editDistance() method is optimized using SIMD (Single Instruction, Multiple Data) instructions to improve performance.
    - It begins by initializing a vector, prev, with the initial values of the dynamic programming table using **_mm256_set_epi32().**
    - Instead of processing one element at a time, the method utilizes **AVX2 (Advanced Vector Extensions 2)** instructions to process eight elements simultaneously, maximizing parallelism and computational efficiency.
    - Within the loop, the strings are loaded into SIMD vectors for comparison using **_mm256_loadu_si256()** to load unaligned memory.
    - SIMD operations such as **_mm256_cmpeq_epi32()** and _mm256_min_epi32() are used to perform comparisons and calculate the minimum value efficiently across the SIMD vectors.
    - The results are updated in the dynamic programming table using SIMD instructions, significantly reducing the number of iterations required to compute the edit distance and leading to faster execution times compared to non-SIMD optimized approaches.

5. **Execution Time Measurement:**
    - The execution time is measured using chrono::steady_clock before and after the edit distance computation.
    - The duration between these two time points is calculated to determine the execution time of the computation.

Compilation and flags:

- **-std=c++11**: This flag sets the C++ language standard to C++11, which enables the compiler to recognize and compile code written according to the C++11 standard, incorporating features introduced in C++11 such as auto type deduction, lambda expressions, and more.
- **-mavx2**: This flag enables **the AVX2 (Advanced Vector Extensions 2)** instruction set, which is an extension to the x86 instruction set architecture. AVX2 introduces wider SIMD (Single Instruction, Multiple Data) registers and additional SIMD operations, allowing the compiler to generate optimized code that can leverage these instructions for parallel processing and potentially improve performance, especially in tasks involving intensive data manipulation.
- **-o Edit_dis_optimized**: With this option, the compiler specifies the name of the output executable file as **Edit_dis_optimized**, allowing the user to customize the name of the compiled program.

```
g++ -std=c++11 -mavx2 -o Edit_dis_optimized Edit_dis_optimized.cpp
```

Proof of Achieved Speedup

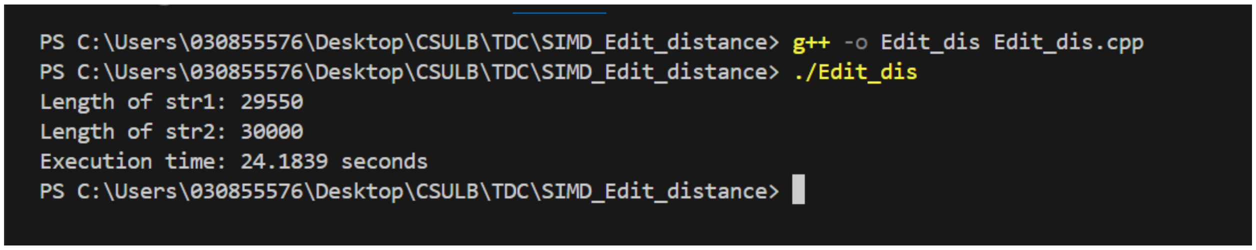
Here, conducted performance measurements on the original and optimized programs, and the results are as follows:

• **Original Code Execution Time:** 24.1839 seconds

• **Optimized Code Execution Time:** 2.58389 seconds

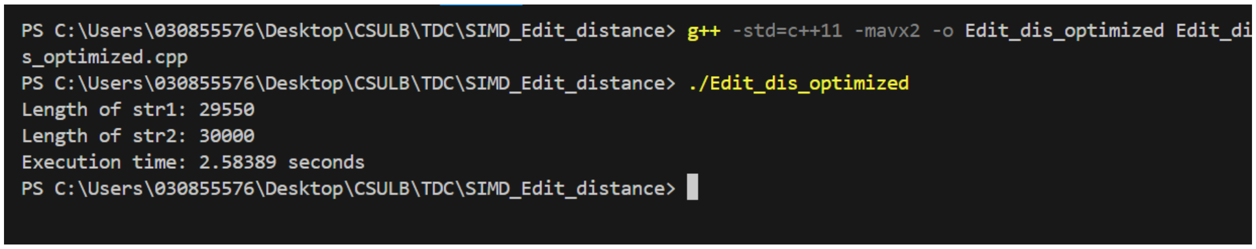
The original code took **24.1839 seconds** to execute, while the optimized code with SIMD achieved a significant speedup, completing in only **2.58389 seconds**. This represents a remarkable improvement **i.e. 9.35x**, demonstrating the effectiveness of SIMD optimization techniques.

Edit_dis.cpp (Original file)



```
PS C:\Users\030855576\Desktop\CSULB\TDC\SIMD_Edit_distance> g++ -o Edit_dis Edit_dis.cpp
PS C:\Users\030855576\Desktop\CSULB\TDC\SIMD_Edit_distance> ./Edit_dis
Length of str1: 29550
Length of str2: 30000
Execution time: 24.1839 seconds
PS C:\Users\030855576\Desktop\CSULB\TDC\SIMD_Edit_distance>
```

Edit_dis_optimized.cpp (SIMD Optimized file)



```
PS C:\Users\030855576\Desktop\CSULB\TDC\SIMD_Edit_distance> g++ -std=c++11 -mavx2 -o Edit_dis_optimized Edit_di
s_optimized.cpp
PS C:\Users\030855576\Desktop\CSULB\TDC\SIMD_Edit_distance> ./Edit_dis_optimized
Length of str1: 29550
Length of str2: 30000
Execution time: 2.58389 seconds
PS C:\Users\030855576\Desktop\CSULB\TDC\SIMD_Edit_distance>
```

Conclusion

In conclusion, the optimization of the Edit distance (**Edit_dis.cpp**) program using SIMD instructions has yielded a substantial improvement in performance. (**Edit_dis_optimized.cpp**). The optimized version demonstrates significantly reduced execution time compared to the original implementation. Leveraging SIMD techniques, particularly **AVX2**, enables parallel processing of data, resulting in enhanced efficiency and faster computation of edit distances. This optimization underscores the importance of utilizing advanced techniques to maximize software performance and highlights the potential for further enhancements in computational efficiency through optimization.